This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2016.2587687, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

$LFSR$-Based Generation of Multicycle Tests

Irith Pomeranz

***Abstract*— This paper describes a procedure for computing a multicycle test set whose scan-in states are compressed into seeds for an $LFSR$, and whose primary input vectors are held constant during the application of a multicycle test. The goal of computing multicycle tests is to provide test compaction that reduces both the test application time and the test data volume. To avoid sequential test generation, the procedure uses a single-cycle test set to guide the computation of multicycle tests. The procedure optimizes every multicycle test, and increases the number of faults it detects, by adjusting its seed, primary input vector, and number of functional clock cycles. Optimizing the seed instead of the scan-in state avoids the computation of scan-in states for which seeds do not exist. Experimental results for benchmark circuits are presented to demonstrate the effectiveness of the procedure.**

***Index Terms*— $LFSR$-based test generation, multicycle tests, test compaction, test data compression.**

I. INTRODUCTION

Between the scan-in and scan-out operations of a test, a single-cycle test has a single functional clock cycle, while a multicycle test has one or more functional clock cycles. Multicycle tests were considered in [1]-[12]. Their effectiveness for test compaction was demonstrated in [1], [2], [9], [11] and [12], and results from the following observations. During a functional clock cycle of a test, the combinational logic of the circuit receives an input pattern that can be used for detecting faults. A larger number of functional clock cycles allows more faults to be detected. As a result, a multicycle test may detect more faults than a single-cycle test. With more detected faults for every test, the number of tests is reduced. This reduces the number of scan operations that a test set requires. With fewer scan operations, the test data volume and test application time are reduced. The fact that each test consists of more functional clock cycles has a negligible effect on the test application time when the number of functional clock cycles is bounded. The test data volume is independent of the number of functional clock cycles if the primary input vector is kept constant during a test. This is a common requirement to address tester limitations that prevent the primary input vector from being changed during a test.

For the discussion in this paper a multicycle test is denoted by $t_i =< p_i, v_i, l_i >$, where p_i is the scan-in state, v_i is the primary input vector, and l_i is the number of functional clock cycles. After p_i is scanned-in, the primary input vector v_i is applied for l_i functional clock cycles. The test ends with a scan-out operation.

The generation of multicycle tests for test compaction requires the number of functional clock cycles l_i in a test to be determined in addition to its scan-in state p_i and its primary input vector v_i. To simplify the test generation procedure, and avoid the need for sequential test generation, it is possible to use a single-cycle test set to guide the generation of a multicycle test set [12]. Thus, if $< q_i, u_i, 1 >$ is a single-cycle test, it is possible to define a multicycle test $< p_i, v_i, l_i >$ by using $p_i = q_i$ and $v_i = u_i$. However, as shown in [12], after this initial assignment, it is important to optimize p_i, v_i and l_i together in order to obtain multicycle tests that detect the largest possible numbers of faults, and thus achieve the highest possible level of test compaction.

The generation of multicycle tests for test compaction becomes more complex when test data compression is used. In one of the commonly used test data compression methods, a test is compressed into a seed for a linear-feedback shift register ($LFSR$) [13]-[24]. The on-chip decompression logic uses the $LFSR$ to apply the test to the circuit. A seed is typically computed based on an incompletely-specified test cube by solving a set of linear equations that relate the bits of the seed with the specified values of the test cube [13]. With this process, optimizing a multicycle test $< p_i, v_i, l_i >$ to increase the number of faults it detects requires a seed to be recomputed after every step that modifies the test, and some modifications of the test cannot be accepted because a seed does not exist for the modified test.

Motivated by these observations, the goal of this paper is to develop a procedure for computing seeds for $LFSR$-based generation of multicycle tests that are effective for test compaction. To avoid sequential test generation, the procedure uses a single-cycle test set similar to [12], and optimizes the multicycle tests to increase the numbers of faults they detect. In contrast to [12], the procedure optimizes the compressed multicycle tests in order to avoid producing tests for which seeds do not exist.

A compressed multicycle test is represented as $t_i =< s_i, v_i, l_i >$, where s_i is a seed that produces the scan-in state p_i of t_i. For simplicity, and since the number of primary inputs is typically significantly smaller than the number of state variables, a seed is computed for the scan-in state p_i. The primary input vector v_i is stored separately. This is consistent with the approach described in [21]. The number of functional clock cycles l_i does not need to be stored for every test if tests with equal numbers of functional clock cycles are stored and applied consecutively.

To achieve the goal of producing compressed multicycle tests that are effective for test compaction, the procedure described in this paper optimizes the seed s_i, the primary input vector v_i, and the number of functional clock cycles l_i together to increase the number of faults that the test detects. By considering the seed s_i directly, the procedure optimizes the scan-in state p_i, and avoids modifications of p_i for which a seed does not exist. Moreover, the single-cycle test set that the procedure uses as guidance does not need to be compressed. To accommodate this case, the procedure initializes the seed s_i randomly, and not based on the scan-in state q_i of a single-cycle test. It is thus possible to use a compact single-cycle test set that is not constrained by the $LFSR$.

The possibility of optimizing a seed s_i was used in [23] to modify seeds that produce fault detection tests into seeds that produce diagnostic tests. The modification of a seed s_i is implemented in [23] by complementing bits of s_i one by one, and recomputing the test t_i that the $LFSR$ produces. A bit complementation is accepted when t_i satisfies certain objectives (in [23] these objectives are related to the generation of diagnostic tests). In the procedure described in this paper, bits of s_i and v_i, as well as the value of l_i, are modified together in order to produce an effective multicycle test.

The target faults in this paper are single stuck-at faults. The procedure is developed assuming that an $LFSR$ is given. The paper also describes a modified binary search process for selecting an $LFSR$ out of a given set of available $LFSR$s.

The paper is organized as follows. Section II describes the computation of a compressed multicycle test $t_i =< s_i, v_i, l_i >$ based on a single-cycle test $w_i =< q_i, u_i, 1 >$. Section III describes the computation of a compact multicycle test set that is made up of compressed multicycle tests based on a single-cycle test set. The modified binary search process for an $LFSR$ is described in Section IV. Section V presents experimental results.

Irith Pomeranz is with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907, U.S.A. (e-mail: pomeranz@ecn.purdue.edu).

II. COMPUTING A COMPRESSED MULTICYCLE TEST

The procedure described in this section accepts a single-cycle test $w_i =< q_i, u_i, 1 >$, a set of target faults F_i, and an initial target L for the number of functional clock cycles in a multicycle test. It produces a compressed multicycle test $t_i =< s_i, v_i, l_i >$ that detects as many faults from F_i as possible.

To check whether w_i is effective in guiding the generation of a multicycle test, the procedure performs fault simulation of F_i under w_i. It stores the set of detected faults in D_i. If $D_i = \emptyset$, the procedure does not attempt to compute a multicycle test based on w_i. It marks that w_i is not effective to avoid considering it again in later iterations. If $D_i \neq \emptyset$, the procedure continues as follows.

Not all the specified values of $w_i =< q_i, u_i, 1 >$ are needed for fault detection. To ensure that only important values guide the generation of t_i, the procedure first changes as many specified values of q_i as possible into unspecified values without losing the detection of any fault from F_i. The remaining specified values are important for the detection of target faults. They can thus be used for guiding the generation of t_i.

For a circuit with k state variables, let $q_i(j)$ be the value of state variable j, where $0 \leq j < k$. For $0 \leq j < k$, if $q_i(j) \neq x$, the procedure assigns $q_i(j) = x$, and simulates D_i under $< q_i, u_i, 1 >$. If all the faults in D_i are detected, the procedure accepts the unspecified value of $q_i(j)$. Otherwise, it restores its previous specified value.

To compute $t_i =< s_i, v_i, l_i >$, the procedure initializes s_i randomly, and assigns $v_i = u_i$ and $l_i = L$. Let p_i be the scan-in state that s_i produces. The procedure simulates F_i under $< p_i, v_i, l_i >$, and stores the number of detected faults in a variable that is denoted by d_{best}. In addition, it computes the Hamming distance between p_i and q_i, and stores it in a variable that is denoted by h_{best}. The Hamming distance is equal to the number of state variables j where $q_i(j) \neq x$ and $p_i(j) \neq q_i(j)$. As t_i is modified, d_{best} stores the largest number of detected faults, and h_{best} stores the smallest Hamming distance obtained with the largest number of detected faults.

The goal of modifying t_i is to increase the number of detected faults (or the value of d_{best}), and reduce the Hamming distance between p_i and q_i (or the value of h_{best}). Increasing the number of detected faults is given a higher priority. If the procedure cannot increase the number of detected faults, reducing the Hamming distance between p_i and q_i may eventually allow t_i to detect faults from D_i.

The modification of t_i is accomplished in three steps that are applied iteratively. The first step attempts to complement bits of s_i. The second step attempts to complement bits of v_i. The third step attempts to replace l_i with a different value from the set $\{1, 2, ..., L_{MAX}\}$, where L_{MAX} is a constant upper bound on l_i.

During the first step, the procedure considers every bit of s_i. With a B-bit $LFSR$, the procedure considers $s_i(j)$ for $0 \leq j < B$. When the procedure considers $s_i(j)$, it complements its value, and recomputes the scan-in state p_i of t_i. It simulates F_i under t_i, and stores the number of detected faults in a variable that is denoted by d_i. In addition, it computes the Hamming distance between p_i and q_i, and stores it in a variable that is denoted by h_i. The procedure accepts the complementation of $s_i(j)$ if $d_i > d_{best}$, or $d_i = d_{best}$ and $h_i < h_{best}$. Thus, to accept the complementation of $s_i(j)$, the procedure requires either an increase in the number of detected faults, or a reduction in the Hamming distance with the same number of detected faults. If this condition is satisfied, the procedure updates d_{best} and h_{best} by assigning $d_{best} = d_i$ and $h_{best} = h_i$. Otherwise, the procedure restores the previous value of $s_i(j)$ by complementing it again.

A similar process is applied to v_i, except that complementing bits of v_i does not affect the Hamming distance between p_i and q_i. The same applies to l_i. For l_i, the procedure considers different numbers of functional clock cycles, which are given by $l_{new} = L_{MAX}$, $L_{MAX} - 1$, ..., 1, in this order. If $l_{new} \neq l_i$, the procedure assigns $l_i = l_{new}$. It simulates F_i under t_i, and stores the number of detected faults in d_i. The procedure accepts the new value of l_i if $d_i \geq d_{best}$. In this case, it assigns $d_{best} = d_i$. Otherwise, it restores l_i to its previous value.

This process prefers a lower value of l_i if it does not reduce the number of detected faults.

The number of iterations of the three steps is a constant that is denoted by N_{MOD}. After N_{MOD} iterations the procedure returns the test t_i, and the number of faults that it detects, d_{best}.

The procedure for computing t_i is summarized next. For uniformity, the Hamming distance between p_i and q_i is considered for s_i, v_i and l_i even though it cannot be affected by modifying v_i or l_i. The number of primary inputs is denoted by n.

Procedure 1: Computing a compressed multicycle test t_i

```
1) Simulate $F_i$ under $< q_i, u_i, 1 >$ and find the set of detected faults, $D_i$. If $D_i = \emptyset$, assign $use(w_i) = 0$, and return $d_{best} = 0$.
2) Unspecify $q_i$ such that $w_i$ would continue to detect all the faults in $D_i$.
3) Specify $s_i$ randomly. Assign $v_i = u_i$ and $l_i = L$.
4) Compute $p_i$. Simulate $F_i$ under $t_i$ and assign the number of detected faults to $d_{best}$. Compute the Hamming distance between $p_i$ and $q_i$, and assign it to $h_{best}$.
5) For $n_{mod} = 0$, 1, ..., $N_{MOD} - 1$:
   a) For $j = 0$, 1, ..., $B - 1$:
      i) Complement $s_i(j)$. Call Procedure $accept\_mod()$. If the procedure returns FALSE, complement $s_i(j)$ again.
   b) For $j = 0$, 1, ..., $n - 1$:
      i) Complement $v_i(j)$. Call Procedure $accept\_mod()$. If the procedure returns FALSE, complement $v_i(j)$ again.
   c) For $l_{new} = L_{MAX}$, $L_{MAX} - 1$, ..., 1, if $l_i \neq l_{new}$:
      i) Assign $l_i = l_{new}$. Call Procedure $accept\_mod()$. If the procedure returns FALSE, restore the previous value of $l_i$.
6) Return $t_i$ and $d_{best}$.
```

Procedure $accept\_mod()$

```
1) Compute $p_i$. Simulate $F_i$ under $t_i$ and assign the number of detected faults to $d_i$.
2) Compute the Hamming distance between $p_i$ and $q_i$, and assign it to $h_i$.
3) If $d_i > d_{best}$, or $d_i = d_{best}$ and $h_i \leq h_{best}$, assign $d_{best} = d_i$ and $h_{best} = h_i$, and return TRUE.
4) Return FALSE.
```

Procedure 1 performs N_{MOD} iterations where it considers B bits of s_i, n bits of v_i, and $L_{MAX} - 1$ options for l_i. In every case it simulates one modified test, for a total of $N_{MOD}(B+n+L_{MAX}-1)$ tests.

III. COMPUTING A COMPRESSED MULTICYCLE TEST SET

This section describes the computation of a compressed multicycle test set based on a single-cycle test set W_1. The test set W_1 is not producible by an $LFSR$ with a limited number of bits. With a bound L_{MAX} on the number of functional clock cycles in a test, the multicycle test set is denoted by $T_{L_{MAX}}$.

TABLE I
EXAMPLE

L	i	spec	l_i	f.c.	i	spec	l_i	f.c.
8	0	51	8	45.65	19	23	3	96.28
	1	39	8	70.67	21	9	6	96.87
	2	35	6	78.28	23	22	1	96.96
	3	21	8	81.49	26	11	1	97.13
	4	22	6	85.21	27	13	5	97.21
	5	20	1	88.08	29	13	1	97.30
	6	8	3	89.52	32	26	4	97.46
	7	28	7	90.87	35	11	7	97.63
	8	4	1	92.05	37	20	6	97.72
	10	15	1	92.48	39	20	1	97.89
	11	21	7	92.98	41	13	2	97.97
	12	15	5	93.66	46	20	1	98.06
	13	5	1	94.00	47	21	1	98.22
	14	6	1	94.42	49	20	1	98.39
	18	27	3	95.69				
7	5	18	5	98.82	32	19	1	99.41
	7	20	1	98.90	39	20	1	99.49
	19	21	1	99.07	41	13	1	99.58
	23	22	1	99.32	46	13	1	99.66
6	39	20	1	99.75				
5	29	13	1	99.83	32	19	1	99.92

The procedure initially assigns $T_{L_{MAX}} = \emptyset$, and includes in a set F all the target faults that are detected by W_1. The procedure constructs $T_{L_{MAX}}$ by performing L_{MAX} iterations over the tests of W_1. The iterations differ in the initial target L for the number of functional clock cycles in a test. The procedure considers $L = L_{MAX}$, $L_{MAX} - 1$, ..., 1 in order to achieve the following goals.

By considering higher values of L earlier, the procedure gives a precedence to the computation of multicycle tests with larger numbers of clock cycles. Such tests allow more target faults to be detected, thus contributing to test compaction. By considering all the values of L down to 1, the procedure ensures that single-cycle tests will be included in $T_{L_{MAX}}$ if this is necessary for detecting some of the faults.

For every value of L, the procedure attempts to compute a test t_i based on every test $w_i =< q_i, u_i, 1 >\in W_1$. If a test t_i is computed, and $d_{best} \neq 0$, the procedure adds t_i to $T_{L_{MAX}}$, and simulates F under t_i with fault dropping.

After considering all the values of L, the procedure performs forward-looking reverse order fault simulation in order to remove unnecessary tests from $T_{L_{MAX}}$.

Several features of the procedure are illustrated by the following example. The example uses a 24-bit primitive $LFSR$ from [25] for ITC-99 benchmark $b07$. The test set W_1 consists of 52 tests, and it achieves a 99.92% single stuck-at fault coverage (the remaining faults are undetectable). The procedure is applied with $L_{MAX} = 8$. Table I shows the multicycle tests that the procedure constructs with L = 8, 7, 6 and 5. The procedure terminates after considering $L = 5$ since all the target faults are detected.

In Table I, column L shows the initial number of clock cycles for a multicycle test. Column i shows the index of the test $w_i \in W_1$ that the procedure uses. Column *spec* shows the number of specified values in the scan-in state q_i of w_i. Column *f.c.* shows the single stuck-at fault coverage after the procedure adds a test based on w_i to T_8.

A multicycle test t_i that the procedure derives with a given value of L may have $l_i \neq L$. The initial value of l_i is L, but the procedure may select a different value. This occurs in Table I for several tests. For example, with $L = 8$, based on $w_2 \in W_1$ the procedure produces a 6-cycle test. Based on $w_5 \in W_1$ the procedure produces a single-cycle test.

The number of specified values in the scan-in state q_i of $w_i \in W_1$ decreases as the fault coverage of the compressed test set increases. A higher fault coverage implies that fewer faults remain to be detected. Therefore, fewer faults are included in F_i, and in the set of faults D_i that w_i is required to detect. With fewer faults in D_i, q_i requires fewer specified values for detecting the faults in D_i. For example, $w_5 \in W_1$ has 20 specified values in its scan-in state after it is unspecified with $L = 8$. The number of specified values decreases to 18 with $L = 7$.

The procedure does not compute a multicycle test based on every single-cycle test. For example, with $L = 8$, the procedure does not generate a test based on w_9, w_{15}, w_{16}, and so on. This results in test compaction.

The procedure is summarized next.

Procedure 2: Computing a multicycle test set

```
1) Let $F$ be the set of target faults that are detected by $W_1$. Assign $use(w_i) = 1$ for every $w_i \in W_1$. Assign $T_{L_{MAX}} = \emptyset$.
2) For $L = L_{MAX}$, $L_{MAX} - 1$, ..., 1, if $use(w_i) = 1$:
    a) For $i = 0, 1, \ldots, |W_1| - 1$:
        i) Call Procedure 1 with the test $w_i =< q_i, u_i, 1 >\in W_1$, $L$ and $F$. If Procedure 1 returns a test $t_i$ and $d_{best} > 0$:
            A) Perform fault simulation with fault dropping of $F$ under $t_i$.
            B) Add $t_i$ to $T_{L_{MAX}}$.
```

Procedure 2 performs L_{MAX} iterations where it calls Procedure 1 at most $|W_1|$ times. Each call to Procedure 1 requires simulation of $N_{MOD}(B+n+L_{MAX}-1)$ tests. Overall, this yields a computational effort that is equivalent to simulation of $O(L_{MAX}|W_1|N_{MOD}(B+n+L_{MAX}-1))$ tests. With constant values for N_{MOD} and L_{MAX}, the number of simulated tests is $O(|W_1|(B+n))$.

IV. SELECTING AN $LFSR$

The $LFSR$ with the smallest number of bits that achieves the fault coverage of W_1, or the highest possible fault coverage, is preferred. This section describes a modified binary search process for such an $LFSR$ out of a given set of available $LFSR$s. The set is denoted by $A = \{\alpha_0, \alpha_1, ..., \alpha_{m-1}\}$. For $0 \leq i < m$, the number of bits in $LFSR$ α_i is denoted by B_i. The $LFSR$s in A are ordered such that $B_0 \leq B_1 \leq ... \leq B_{m-1}$.

In general, an $LFSR$ with a larger number of bits has the potential to yield a higher fault coverage. However, this is not guaranteed. The modified binary search process takes into consideration that, with $B_{i0} < B_{i1}$, the $LFSR$ α_{i0} can achieve a higher fault coverage than α_{i1}. Therefore, considering α_{i1} and finding that it does not achieve the fault coverage of W_1 should not immediately preclude $LFSR$s with indices $i_0 < i_1$ from consideration. This is incorporated into the modified binary search process as follows.

Initially, $i_{lo} = 0$ and $i_{hi} = m - 1$ are the bounds of the modified binary search process. In an arbitrary step, Procedure 2 is applied using α_i where $i = (i_{lo} + i_{hi})/2$. Based on the fault coverage, the bounds i_{lo} and i_{hi} are updated as follows.

(1) If α_i achieves the fault coverage of W_1, $i_{hi} = i - 1$ is assigned. In this case, the binary search continues to search among the $LFSR$s with the lower numbers of bits as in the conventional case.

(2) If α_i does not achieve the fault coverage of W_1, in a conventional binary search process, $i_{lo} = i + 1$ is assigned in order to continue the search among the $LFSR$s with the higher numbers of bits. To allow $LFSR$s with lower numbers of bits to be considered as well, the modified binary search process assigns $i_{lo} = (i_{lo} + i)/2$. If this does not increase i_{lo}, then $i_{lo} = i_{lo} + 1$ is assigned.

Table II illustrates the modified binary search process for IWLS-05 benchmark $i2c$. The test set W_1 achieves 100% single stuck-at fault coverage. The set of available $LFSR$s consists of 61 $LFSR$s with numbers of bits between 4 and 64. Initially, $i_{lo} = 0$ and

TABLE II

MODIFIED BINARY SEARCH

i_{lo}	i_{hi}	i	B	f.c.
0	60	30	34	99.87
15	60	37	41	99.83
26	60	43	47	99.96
34	60	47	51	100.00
34	46	40	44	99.96
37	46	41	45	100.00
37	40	38	42	99.91
38	40	39	43	100.00

TABLE III

EXPERIMENTAL RESULTS

				func					
circuit	B	L	tests	max	ave	cycles	bits	f.c.	ntime
s1423	74	1	38	1	1.00	2924	2812	99.08	-
s1423	23	8	27	8	4.11	2183	621	99.08	1523.90
s1423	23	1	47	1	1.00	3599	1081	98.94	363.10
s1423	23	8	31	8	3.52	2477	713	98.61	667.83
s5378	179	1	111	1	1.00	20159	19869	99.13	-
s5378	47	8	154	8	1.59	27990	7238	99.13	774.14
s5378	47	1	168	1	1.00	30419	7896	99.13	275.01
s5378	47	8	181	8	1.60	32868	8507	98.94	1673.43
s9234	228	1	143	1	1.00	32975	32604	93.47	-
s9234	94	8	163	8	2.36	37776	15322	93.47	3595.50
s9234	94	1	181	1	1.00	41677	17014	93.47	1050.82
s9234	94	8	239	8	3.14	55470	22466	91.74	3789.69
s13207	669	1	238	1	1.00	160129	159222	98.46	-
s13207	68	8	212	8	2.43	143012	14416	98.46	1515.47
s13207	68	1	302	1	1.00	203009	20536	98.46	575.85
s13207	68	8	295	8	2.41	198734	20060	98.09	3345.53
s15850	597	1	118	1	1.00	71161	70446	96.68	-
s15850	79	8	240	8	2.13	144388	18960	96.68	2226.16
s15850	79	1	286	1	1.00	171625	22594	96.67	573.72
s15850	79	8	252	8	2.33	151628	19908	94.33	3780.59
s35932	1728	1	20	1	1.00	36308	34560	89.81	-
s35932	4	8	15	8	4.80	27720	60	89.81	326.24
s35932	4	1	33	1	1.00	58785	132	89.39	207.82
s35932	4	8	16	8	3.62	29434	64	89.81	309.29
b04	66	1	44	1	1.00	3014	2904	99.85	-
b04	15	8	39	6	1.69	2706	585	99.85	382.58
b04	15	1	40	1	1.00	2746	600	99.85	99.53
b04	15	8	37	8	1.70	2571	555	99.63	432.46
b07	51	1	52	1	1.00	2755	2652	99.92	-
b07	24	8	38	8	3.13	2108	912	99.92	385.56
b07	24	1	57	1	1.00	3015	1368	99.75	121.94
b07	24	8	33	8	3.79	1859	792	99.15	410.95
b14	247	1	332	1	1.00	82583	82004	94.87	-
b14	123	8	265	7	1.56	66115	32595	94.87	965.11
b14	123	1	317	1	1.00	78863	38991	94.87	367.33
b14	123	8	152	8	2.63	38191	18696	90.24	3417.93
des_area	128	1	118	1	1.00	15350	15104	100.00	-
des_area	4	8	68	8	2.62	9010	272	100.00	3362.74
des_area	4	1	77	1	1.00	10061	308	100.00	731.21
des_area	4	8	104	8	2.18	13667	416	100.00	3818.72
i2c	128	1	45	1	1.00	5933	5760	100.00	-
i2c	43	8	50	8	2.30	6643	2150	100.00	1040.32
i2c	43	1	69	1	1.00	9029	2967	99.53	340.01
i2c	43	8	59	8	2.88	7850	2537	99.27	1302.63
pci_spoci_ctrl	60	1	146	1	1.00	8966	8760	99.94	-
pci_spoci_ctrl	27	8	104	8	2.42	6552	2808	87.25	3333.54
pci_spoci_ctrl	27	1	113	1	1.00	6953	3051	83.46	1405.87
pci_spoci_ctrl	27	8	97	8	2.33	6106	2619	77.75	5515.83
sasc	117	1	22	1	1.00	2713	2574	100.00	-
sasc	8	8	25	8	3.72	3135	200	100.00	516.08
sasc	8	1	36	1	1.00	4365	288	99.88	133.58
sasc	8	8	23	8	3.57	2890	184	100.00	510.15
simple_spi	131	1	36	1	1.00	4883	4716	100.00	-
simple_spi	39	8	31	8	2.90	4282	1209	100.00	961.51
simple_spi	39	1	44	1	1.00	5939	1716	99.90	303.20
simple_spi	39	8	35	8	3.71	4846	1365	99.14	1121.72
spi	229	1	406	1	1.00	93609	92974	99.98	-
spi	83	8	260	8	1.80	60238	21580	99.98	1111.36
spi	83	1	373	1	1.00	86019	30959	99.97	423.95
spi	83	8	252	8	1.97	58433	20916	99.80	1233.31
systemcdes	190	1	79	1	1.00	15279	15010	100.00	-
systemcdes	4	8	23	8	7.00	4721	92	100.00	930.44
systemcdes	4	1	71	1	1.00	13751	284	99.00	915.54
systemcdes	4	8	26	8	6.12	5289	104	100.00	876.98
tv80	359	1	489	1	1.00	176399	175551	99.33	-
tv80	89	8	324	8	3.42	117782	28836	99.33	1158.32
tv80	89	1	469	1	1.00	169199	41741	99.29	331.65
tv80	89	8	401	8	3.59	145756	35689	98.82	1953.81
usb_phy	98	1	32	1	1.00	3266	3136	100.00	-
usb_phy	20	8	29	8	2.90	3024	580	100.00	590.00
usb_phy	20	1	38	1	1.00	3860	760	100.00	166.00
usb_phy	20	8	30	8	3.13	3132	600	100.00	692.00
wb_dma	523	1	66	1	1.00	35107	34518	100.00	-
wb_dma	126	8	111	8	1.54	58747	13986	100.00	13085.93
wb_dma	126	1	130	1	1.00	68643	16380	99.99	4000.95
wb_dma	126	8	143	8	1.63	75545	18018	99.63	8651.51

$i_{hi} = 60$. With these values, Procedure 2 is applied using $i = 30$. The corresponding $LFSR$ has $B_{30} = 34$ bits. The fault coverage is 99.87%, lower than the fault coverage of W_1. In a conventional binary search process, this would result in $i_{lo} = 31$. In the modified process, $i_{lo} = (0 + 30)/2 = 15$. With $i_{lo} = 15$ and $i_{hi} = 60$, Procedure 2 is applied using $i = 37$. The fault coverage is again lower than the fault coverage of W_1. This results in $i_{lo} = (15 + 37)/2 = 26$. The search continues as shown in Table II. The search yields α_{39} as the best $LFSR$. This value would not be obtained in a conventional search after α_{43} yields a fault coverage that is lower than that of W_1.

V. EXPERIMENTAL RESULTS

This section presents the results of Procedure 2 for single stuck-at faults in benchmark circuits.

The test set W_1 is a compact test set that was produced by the procedure from [26] with the target of detecting each detectable single stuck-at fault once.

Procedure 2 was applied using $L_{MAX} = 8$. This value is high enough to demonstrate the advantages of multicycle tests. A test is modified during $N_{MOD} = 4$ iteration. Smaller numbers of iterations are typically sufficient.

The set of available $LFSR$s consists of primitive $LFSR$s from [25]. All the B-bit $LFSR$s with $4 \leq B \leq 128$, and several of the $LFSR$s with $129 \leq B \leq 300$, are available. For a circuit with k state variables, $B \leq k/2$.

For comparison, Procedure 2 is applied with $L_{MAX} = 1$ and the $LFSR$ that is selected for $L_{MAX} = 8$. With $L_{MAX} = 1$, Procedure 2 is allowed to perform eight iterations, as in the case where $L_{MAX} = 8$, by repeating Step 2 eight times with $L = 1$.

To demonstrate the advantages of the bit complementation process used by Procedure 1 over a random search process, a random version of Procedure 1 was implemented and used as part of Procedure 2. The modified procedures are referred to as Procedure 1R and 2R, respectively. Procedure 1R selects a new random seed s_i or primary input vector v_i every time Procedure 1 complements a bit of s_i or v_i, respectively. In this way, the procedures consider the same numbers of seeds and primary input vectors, but they are random in Procedure 1R. Procedure 2R is identical to Procedure 2 except that it calls Procedure 1R instead of Procedure 1. Procedure 2R was run with $L_{MAX} = 8$ and the $LFSR$ selected for Procedure 2 with $L_{MAX} = 8$. The test set obtained by Procedure 2R is referred to as R_8.

The results are given in Table III as follows. The first row for every circuit describes W_1, for which scan-in states cannot be produced by an $LFSR$ with a limited number of bits. The second row describes the multicycle test set T_8 produced by Procedure 2 with the $LFSR$ selected by the modified binary search process. The third row describes the single-cycle test set T_1 produced by Procedure 2 with the same $LFSR$. The fourth row describes the multicycle test set R_8 produced by Procedure 2R using a random search process and the same $LFSR$.

Column B shows the number of $LFSR$ bits. For W_1, $B = k$. Column L shows the value of L_{MAX}. Column $tests$ shows the number of tests in the test set. Column $func$ shows the maximum

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2016.2587687, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

and average number of functional clock cycles in a test. Column $cycles$ shows the number of clock cycles required for applying the test set, including scan and functional clock cycles. Column $bits$ shows the number of bits required for storing scan-in states or seeds. Column $f.c.$ shows the single stuck-at fault coverage. Column $ntime$ shows the run time normalized to the run time for single stuck-at fault simulation of W_1.

The following points are important when considering the results shown in Table III. The main purpose of using an $LFSR$ with a limited number of bits is to achieve test data compression. From Table III it can be observed that the number of bits required for storing the scan-in states of T_1 is lower than that of W_1. The number of bits required for storing the scan-in states of T_8 is lower than for T_1 and W_1. Thus, the use of multicycle tests strengthens the ability to achieve test data compression.

The requirement to produce a test set by an $LFSR$ with a limited number of bits may limit the fault coverage and increase the number of tests. Therefore, the number of clock cycles required for applying the test set may increase. Nevertheless, for most of the circuits considered, T_8 achieves the fault coverage of W_1. Without using multicycle tests, and for the same number of $LFSR$ bits, the fault coverage of T_1 is sometimes lower than that of W_1 and T_8.

In addition, there are several cases where the number of clock cycles required for applying T_8 is lower than that of W_1. The number of clock cycles required for the application of T_8 is lower than for T_1 even for a higher fault coverage. This is consistent with the ability of multicycle tests to provide test compaction.

In most of the cases considered, when a random search process is used instead of the bit complementation process of Procedure 1, the fault coverage is lower. There is only one case ($sasc$) where the fault coverage is the same and the random search process produces a lower number of bits.

Overall these results demonstrate that the advantages of multicycle tests in achieving test compaction are valid when the tests are required to be producible by an $LFSR$ with a limited number of bits in order to achieve test data compression.

VI. CONCLUDING REMARKS

This paper described a procedure for computing a multicycle test set with the following properties. (1) The scan-in states are compressed into seeds for an $LFSR$. (2) The primary input vectors are held constant during the application of a multicycle test. The procedure is guided by a single-cycle test set. This test set does not have to be applicable using an $LFSR$ with a limited number of bits. The procedure adjusts an initially random seed, the primary input vector, and the number of functional clock cycles of each multicycle test to detect the largest possible number of faults. This process is guided by a single-cycle test. Experimental results for benchmark circuits demonstrated the effectiveness of multicycle tests in achieving test compaction when the tests are required to be producible by an $LFSR$ in order to achieve test data compression.

REFERENCES

[1] S. Y. Lee and K. K. Saluja, "Test Application Time Reduction for Sequential Circuits with Scan", IEEE Trans. on Computer-Aided Design, Sept. 1995, pp. 1128-1140.

[2] I. Pomeranz and S. M. Reddy, "Static Test Compaction for Scan-Based Designs to Reduce Test Application Time", in Proc. Asian Test Symp., 1998, pp. 198-203.

[3] P. C. Maxwell, R. C. Aitken, K. R. Kollitz and A. C. Brown, "IDDQ and AC Scan: The War Against Unmodelled Defects", in Proc. Intl. Test Conf., 1996, pp. 250-258.

[4] J. Rearick, "Too Much Delay Fault Coverage is a Bad Thing", in Proc. Intl. Test Conf., 2001, pp. 624-633.

[5] X. Lin and R. Thompson, "Test Generation for Designs with Multiple Clocks", in Proc. Design Autom. Conf., 2003, pp. 662-667.

[6] G. Bhargava, D. Meehl and J. Sage, "Achieving Serendipitous N-Detect Mark-Offs in Multi-Capture-Clock Scan Patterns", in Proc. Intl. Test Conf, 2007, Paper 30.2.

[7] I. Park and E. J. McCluskey, "Launch-on-Shift-Capture Transition Tests", in Proc. Intl. Test Conf., 2008, pp. 1-9.

[8] E. K. Moghaddam, J. Rajski, S. M. Reddy and M. Kassab, "At-Speed Scan Test with Low Switching Activity", in Proc. VLSI Test Symp., 2010, pp. 177-182.

[9] I. Pomeranz, "Generation of Multi-Cycle Broadside Tests", IEEE Trans. on Computer-Aided Design, Aug. 2011, pp. 1253-1257.

[10] I. Pomeranz, "Multi-Cycle Tests with Constant Primary Input Vectors for Increased Fault Coverage", IEEE Trans. on Computer-Aided Design, Sep. 2012, pp. 1428-1438.

[11] I. Pomeranz, "Multi-Cycle Broadside Tests with Runs of Constant Primary Input Vectors", IET Computers & Digital Techniques, March 2014, pp. 90-96.

[12] I. Pomeranz, "A Multi-Cycle Test Set Based on a Two-Cycle Test Set with Constant Primary Input Vectors", IEEE Trans. on Computer-Aided Design, July 2015, pp. 1124-1132.

[13] B. Koenemann, "LFSR-Coded Test Patterns for Scan Designs", in Proc. Europ. Test Conf., 1991, pp. 237-242.

[14] S. Hellebrand, S. Tarnick, J. Rajski and B. Courtois, "Generation of Vector Patterns Through Reseeding of Multiple-Polynomial Linear Feedback Shift Register", in Proc. Intl. Test Conf., 1992, pp. 120-129.

[15] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, B. Keller and B. Koenemann, "OPMISR: The Foundation for Compressed ATPG Vectors", in Proc. Intl. Test Conf., 2001, pp. 748-757.

[16] J. Rajski, J. Tyszer, M. Kassab, N. Mukherjee, R. Thompson, K.-H. Tsai, A. Hertwig, N. Tamarapalli, G. Mrugalski, G. Eide and J. Qian, "Embedded Deterministic Test for Low Cost Manufacturing Test", in Proc. Intl. Test Conf., 2002, pp. 301-310.

[17] A.-W. Hakmi, H.-J. Wunderlich, C.G. Zoellin, A. Glowatz, F. Hapke, J. Schloeffel and L. Souef, "Programmable deterministic Built-In Self-Test", in Proc. Intl. Test Conf., 2007, pp. 1-9.

[18] D. Czysz, G. Mrugalski, N. Mukherjee, J. Rajski, P. Szczerbicki and J. Tyszer, "Deterministic Clustering of Incompatible Test Cubes for Higher Power-Aware EDT Compression", IEEE Trans. on Computer-Aided Design, Aug. 2011, pp. 1225-1238.

[19] A. Chandra, J. Saikia and R. Kapur, "Breaking the Test Application Time Barriers in Compression: Adaptive Scan-Cyclical (AS-C)", in Proc. Asian Test Symp, 2011 , pp. 432-437.

[20] O. Acevedo and D. Kagaris, "Using the Berlekamp-Massey Algorithm to Obtain LFSR Characteristic Polynomials for TPG", in Proc. Intl. Symp. on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, 2012, pp. 233-238.

[21] X. Lin and J. Rajski, "On Utilizing Test Cube Properties to Reduce Test Data Volume Further", in Proc. Asian Test Symp., 2012, pp. 83-88.

[22] T. Moriyasu and S. Ohtake, "A Method of One-Pass Seed Generation for LFSR-Based Deterministic/Pseudo-Random Testing of Static Faults", in Proc. Latin-American Test Symp., 2015, pp. 1-6.

[23] I. Pomeranz, "Computation of Seeds for $LFSR$-Based Diagnostic Test Generation", IEEE Trans. on Computer-Aided Design, Dec. 2015, pp. 2004-2012.

[24] I. Pomeranz, "Computing Seeds for $LFSR$-Based Test Generation from Non-Test Cubes", IEEE Trans. on VLSI Systems, 2016.

[25] P. H. Bardell, W. H. McAnney and J. Savir, $Built-In\ Test\ for\ VLSI\ Pseudorandom\ Techniques$, Wiley Interscience, 1987.

[26] I. Pomeranz and S. M. Reddy, "Forming N-Detection Test Sets Without Test Generation", ACM Trans. on Design Automation, No. 2, 2007, Article 18.